МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивные алгоритмы

Вариант 23

Студент гр. 8304 _____ Барышев А.А.

Преподаватель _____ Фирсов К.В.

Санкт-Петербург

2019

Цель работы.

Разработать программу, которая, имея на входе заданное

логическое_выражение, не содержащее вхождений

идентификаторов, вычисляет значение этого выражения и печатает само выражение и его значение.
Определение логического выражения:
логическое_выражение ::= true|false |NOT(операнд) | операция (операнды)
операция::= AND | OR
операнды::= операнд|операнд, операнды
операнд::=логическое_выражение

Описание программы.

Данная программа использует рекурсивный алгоритм для вычисления значения логического выражения. Рекурсия позволяет определять, какую операцию можно вычислить сразу(то есть такую операцию, вычисление которой не требует вычисления других операций), определяет позицию этой операции в строке. Далее данная часть строки заменяется на результат действия операции на данные операнды. Программа снова определяет последнюю операцию, снова заменяет часть строки на результат. Так до тех пор, пока строка не будет приведена к виду «true» или «false».

Функции.

Основные функции данной программы:

```
void  LastOperation(char* string, int position, int last_position, int last_operation);
void calculate(char* string, int operation_pos);
calculate_operation(char* logst);
```

Функция LastOperation находит операцию, вычислить которую можно без вычисления других операций, т. е. если есть логическое_выражение вида: and(true,not(false)), то функция LastOperation вызовет функцию calculate, передав туда позицию, с которой начинается операция not(false). Далее функция calculate из строки логического выражения берёт строку с операцией(в нашем примере это будет not(false)) и помещает в функцию calculate_operation. Последняя функция работает по следующему принципу: проверяется подстрока до символа “(“, с помощью функции strcmp определяется, какая операция в подстроке. Далее вычисляется значение, которое вернёт операция. Для операции not: проверяется первая буква операнда. Если она соответствует «f», то, поскольку все логические выражения строго соответствуют определению, программа идентифицирует это, как not(false), и возвращает true. По аналогичному принципу работает вычисление операций and и or.

Тесты.

Данная программа тестировалась на разных наборах данных, начиная от строк, содержащих только один операнд, заканчивая строками с большим количеством вложенных операций. Текстовый документ с результатами работы программы формируется автоматически. Результат работы программы:

Logic statement: and(not(false),or(false,false,false,not(true),true),not(false),true)
Next step: and(not(false),or(false,false,false,not(true),true),true,true)
Next step: and(not(false),or(false,false,false,false,true),true,true)
Next step: and(not(false),true,true,true)
Next step: and(true,true,true,true)
Next step: true
RESULT: true

Logic statement: or(false,false,false,false,false)
Next step: false
RESULT: false
Logic statement: and(true,true,true,true)
Next step: true
RESULT: true

Logic statement: not(false)
Next step: true
RESULT: true

Logic statement: true
RESULT: true

Logic statement: false
RESULT: false

Logic statement: not(true)
Next step: false
RESULT: false

Logic statement: and(true,false)
Next step: false
RESULT: false

Logic statement: not(and(or(true,(and(or(and(not(true),not(false),true)))))))

Next step: not(and(or(true,(and(or(and(not(true),true,true)))))))

Next step: not(and(or(true,(and(or(and(false,true,true)))))))

Next step: not(and(or(true,(and(or(false))))))

Next step: not(and(or(true,(and(false)))))

Next step: not(and(or(true,(false))))

Next step: not(and(true)))

Next step: not(true))

Next step: false

RESULT: false

Logic statement:
or(not(true),false,and(true,and(false,and(true,or(false,true,false)))),false,and(true,true,false),not(true),not(false))

Next step:
or(not(true),false,and(true,and(false,and(true,or(false,true,false)))),false,and(true,true,false),not(true),true)

Next step:
or(not(true),false,and(true,and(false,and(true,or(false,true,false)))),false,and(true,true,false),false,true)

Next step: or(not(true),false,and(true,and(false,and(true,or(false,true,false)))),false,false,false,true)

Next step: or(not(true),false,and(true,and(false,and(true,true))),false,false,false,true)

Next step: or(not(true),false,and(true,and(false,true)),false,false,false,true)

Next step: or(not(true),false,and(true,false),false,false,false,true)

Next step: or(not(true),false,false,false,false,false,true)

Next step: or(false,false,false,false,false,false,true)

Next step: true

RESULT: true

Logic statement: or(false,false,false,false,false,false,false,not(false))

Next step: or(false,false,false,false,false,false,false,true)

Next step: true

RESULT: true

Также в программе предусмотрена возможность ввода данных с консоли/файла, вывода в файл/на консоль. Это сделано исключительно для удобства и универсальности тестирования.

Выводы.

Был получен опыт использования рекурсивных алгоритмов.

КОД ПРОГРАММЫ.

```
#include <iostream>
#include <cstring>
#include <cstdlib>

using namespace std;

#define STREAM reading //reading from the file if "reading", reading from conslole if "stdin"

const char Term[] = "END";
char array[5][10] = {"false", "true", "not", "or", "and"};
bool EnterInFile = true; //output in the file or in conslole
FILE *writing;

void report(char* logst, const char ReportString[]){
	if(EnterInFile){
		fprintf(writing, "%s: ", ReportString);
		fprintf(writing, "%s\n", logst);
	}
	else{
		printf("%s: ", ReportString);
		printf("%s\n", logst);
	}
}

void copy_string(char* str1, char* str2){
	for(int i = 0; i < (int)strlen(str2); i++)
		str1[i] = str2[i];
	str1[strlen(str2)] = '\0';
}

int calculate_operation(char* logst){
	char buf[10];
	int count = 0;
	for(int i = 0; i < (int)strlen(logst); i++){
		buf[i] = logst[i];
		count++;
		if(logst[i + 1] == '('){
			buf[i + 1] = '\0';
			break;
		}
```

```
        }
        if(strcmp(buf, array[2]) == 0){
                for(int i = count + 1; i < (int)strlen(logst); i++){
                        if(logst[i] == 'f')
                                return 1;
                        else
                                return 0;
                }
        }
        if(strcmp(buf, array[3]) == 0){
                for(int i = count + 1; i < (int)strlen(logst); i++){
                        if(logst[i] == 't'){
                                return 1;
                        }
                }
                return 0;
        }
        if(strcmp(buf, array[4]) == 0){
                for(int i = count + 1; i < (int)strlen(logst); i++){
                        if(logst[i] == 'f'){
                                return 0;
                        }
                }
                return 1;
        }
        return 0;
}

void put_string(char* str1, char* str2, int position, int term_simbol_pos){
        int count = 0;
        for(int i = position; i < (int)strlen(str2) + position; i++){
                str1[i] = str2[count];
                count++;
        }
        count = 0;
        for(int i = strlen(str2) + position; i < (int)strlen(str1); i++){
                str1[i] = str1[term_simbol_pos + count];
                count++;
                if(str1[term_simbol_pos + count] == '\0'){
                        str1[i + 1] = '\0';
                        break;
                }
```

```
        }
}

void calculate(char* string, int operation_pos){
        if(operation_pos != 0){
                char buf_last_operation[strlen(string)];
                int count = 0;
                for(int i = operation_pos + 1; i < (int)strlen(string); i++){
                        cout << string[i];
                        buf_last_operation[count] = string[i];
                        count++;
                        if(string[i] == ')')
                                break;
                }
                buf_last_operation[count] = '\0';
                if(calculate_operation(buf_last_operation)){
                        copy_string(buf_last_operation, array[1]);
                }
                else
                        copy_string(buf_last_operation, array[0]);
                put_string(string, buf_last_operation, operation_pos + 1, operation_pos + 1 + count);
        }
        else{
                if(calculate_operation(string)){
                        copy_string(string, array[1]);
                }
                else
                        copy_string(string, array[0]);
        }
        cout << endl;
}

void  LastOperation(char* string, int position, int last_position, int last_operation){
        char buf[30];
        int count = 0;
        int delta = 0;
        if(position != 0)
                last_position = position - 1;
        else
                last_position = position;
        for(int i = position; i < (int)strlen(string); i++){
                if(string[i] == '(' or string[i] == ',' or string[i] == ')'){
```

```
                        if(string[i + 1] == '\0'){
                                delta++;
                                break;
                        }
                        while(string[i + 1] == '(' or string[i + 1] == ',' or string[i + 1] == ')'){
                                delta++;
                                i++;
                                if(string[i + 1] == '\0'){
                                        delta++;
                                        break;
                                }
                        }
                        break;
                }
                else{
                        buf[count] = string[i];
                        count++;
                }
        }
        if(position < (int)strlen(string)){
                position = position + count + delta;
                buf[count] = '\0';
        }
        if(strcmp(string, array[0]) == 0 or strcmp(string, array[1]) == 0){
                        report(string, "RESULT");
                        return;
        }
        else if(count == 0){
                calculate(string, last_operation);
                report(string, "Next step");
                LastOperation(string, 0, 0, 0);
        }
        else if(strcmp(buf, array[0]) == 0 or strcmp(buf, array[1]) == 0){
                LastOperation(string, position + 1, last_position, last_operation);
        }
        else if(strcmp(buf, array[2]) == 0 or strcmp(buf, array[3]) == 0 or strcmp(buf, array[4]) == 0){
                LastOperation(string, position + 1, last_position, last_position);
        }
}

void MemForString(char** string, int count){
        string[count - 1] = (char*)malloc(500 * sizeof(char));
```

```
}

void Dellb(char* string){
	if(string[strlen(string) - 1] == '\n' )
		string[strlen(string) - 1] = '\0';
}

int main(){
	int count = 1;

	FILE *reading;
	reading = fopen("Tests/input.txt", "r");
	writing = fopen("result.txt", "w");

	cout << "Enter the logical sentences."
	<< "\nYou may not start with the next symbols: "
	<< "space, tab and transfer.\nTerminal word is \"END\"\n";

	char** string = (char**)malloc(count * sizeof(char*));
	MemForString(string, count);
	fgets(string[count - 1], 500, STREAM);
	Dellb(string[count - 1]);
	while(strcmp(string[count - 1], Term) != 0){
		report(string[count - 1], "Logic statement");
		LastOperation(string[count - 1], 0, 0, 0);
		count++;
		string = (char**)realloc(string, count * sizeof(char*));
		MemForString(string, count);
		fgets(string[count - 1], 500, STREAM);
		Dellb(string[count - 1]);
	}

	fclose(reading);
	fclose(writing);

	for(int i = 0; i < count; i++)
		free(string[i]);
	free(string);

	return 0;
}
```